CODE LOGIC *BY SIDDHARTH ASHUTOSH*

BOOKING SERVICE:
Folder Name: booking

1. MySql database is used to store all the records.
2. Table *hotelbooking* is created according to the given schema. The below configuration is provided in application.properties.
   spring.datasource.url=jdbc:mysql://localhost:3306/hotelbooking
3. BookingInfoEntity class under the Entities folder contains the schema of the table.
4. Dto, Dao folders have been created respectfully to access predefined queries and convert response objects to entity type.
5. Booking service contains the below methods:
   A) *calculateRoomPrice*: It takes the Booking entity type and sets the room price in the paytable based on the calculation.
   B) *getRandomNumbers* : generates a list of random room numbers.
   C) *getCurrentDate*: sets the current date in the table.
   D) updateTransactionId: updates the transactionId after receiving a successful response from the payment service.
   E) *findBookingId*: checks if the id exists in the table. If not, it throws a RecordNotFoundException(“Invalid Booking Id”).
   F) *checkPaymentMode*: if the payment mode is wrong, it throws a RecordNotFoundException (“Invalid mode of payment”);
   G) *addBooking* : It saves the data in the table after doing all the operations.
6. The controller has below API endpoints:
   A) *http://localhost:8081/hotel/booking*:
   1. The request body is converted to BookingDTO type. With the use of model mapper, bookingDTO is converted to corresponding entity type(Booking).
   2. bookingService.addBooking(booking) from services is called that calculates the room price, generates random rooms, assigns the bookedOn date and saves the entry in the database.
   B) http://localhost:8081/booking/{bookingId}/transaction:
   1. When this URI is called, the object is converted into paymentDTO type and findBookingId and checkPaymentMode of bookingService throws exceptions if the *bookingId* doesn't exist in the table or the paymentMode is neither “CARD” or “UPI”.
   2. If the *bookingId* and paymentMode are correct, an API call to the payment service is made, which returns the transactionId. The API URI is stored in the application.properties as *payment.url =http://localhost:8083/payment/transaction*
   3. This transactionId is updated in the hotelbooking table by calling bookingService.updateTransactionId method.

<u>PAYMENT SERVICE:</u>

Folder Name: payment

7. MySql database is used to store all the records.
8. Table *hoteltransaction* is created according to the given schema. The below configuration is provided in application.properties.
   spring.datasource.url=jdbc:mysql://localhost:3306/hoteltransaction
9. Payment class under the Entities folder contains the schema of the table.
10. Dto, Dao folders have been created respectfully to access predefined queries and convert response objects to entity type.
11. Payment service contains the below methods:
    H) *generateTransaction*: It saves the payment details in the table and returns the transactionId.
    I) *getPaymentDetails*: returns the payment details by taking transaction Id as its argument.
12. The controller has below API endpoints:
    C) *http://localhost:8083/payment/transaction*:
    1. The request body is converted to PaymentDTO type. With the use of model mapper, PaymentDTO is converted to corresponding entity type(Payment).
    2. paymentService.generateTransaction(payment) from services is called that saves the entry in the database and returns the generated transaction Id to the booking service.

    D) http://localhost:8083/payment/transaction/{transactionId}:
    1. When this URI is called, paymentService.getPaymentDetails(transactionId) method from the payment service returns the payment details corresponding to the transactionId.

<u>Eureka Server:</u>

Folder Name: Service registry

Booking service and Payment service are registered on eureka server with *BOOKING-SERVICE* and *PAYMENT-SERVIC*E names respectively. The eureka server is running on 8761 port. Both the service will be visible running on the server.